A Framework for Policy Based Secure Intra Vehicle Communication

Mohammad Hamad*, Marcus Nolte†, Vassilis Prevelakis*
*Institute of Computer and Network Engineering, TU Braunschweig
{mhamad,prevelakis}@ida.ing.tu-bs.de
†Institute of Control Engineering, TU Braunschweig
nolte@ifr.ing.tu-bs.de

***Abstract*—Over the past two decades, significant developments were introduced within the vehicular domain, evolving the modern vehicle into a network of dozens of embedded systems each hosting one or more applications. Communications within this distributed environment while adhering to safety-critical and secure systems guidelines implies the formulation of a comprehensive and consistent communications policy. Creating this policy is a complex, error-prone and labor-intensive task, requiring detailed knowledge of possible communication paths between all possible components of the system. For this reason, it is often skipped, trusting that each task will behave as intended and interact only with its peers. Traditional testing provides sufficient confidence to allow certification. Nevertheless, the existing process ignores malicious interference, whereby an adversary compromises a low-criticality process or subsystem and uses that to attack other subsystems, effectively taking over the vehicle. In this paper, we propose a framework to build a secure communications policy gradually by integrating it through the design and life cycle of vehicle's software components. We also propose a security module which acts as a connection policy checker vetting the incoming and outgoing communications and enforcing the distributed security policy.**

I. Introduction

In recent years, E/E systems in the automotive domain have become increasingly complex [1]. Driven e.g. by the integration of advanced driver assistance systems, electronic instrument clusters or entertainment systems, modern vehicles rely on a distributed embedded system. A modern car contains more than 70 microcontroller-based computers [2], known as electronic control units (ECUs). These ECUs are distributed throughout the vehicle and are interconnected using different bus systems such as CAN, MOST or FlexRay. Recent trends also show a tendency to integrate Ethernet-based communications [3]. More than 100 millions Lines Of software Code (LOC) [4] are integrated into a premium vehicle to control different functions, which are ranging from mundane such as controlling courtesy lights to highly critical applications such as engine control.

Within the vehicle, multiple ECUs typically share a common bus. These sub networks are often connected to other parts of the vehicle networks via gateways. This kind of network architecture is supposed to satisfy the complex interaction requirements between the different systems. The high degree of interconnectivity in the vehicle is not only driven by the desire for safety (e.g. by connecting multiple sensors in different parts of the vehicle to processing ECUs), but also by comfort considerations (e.g. in case an audio system needs to read the vehicle's velocity to adjust the sound volume). On the other hand, the unrestricted interaction between components of heterogeneous criticality may also create vulnerabilities from a security perspective [5].

The traditional approach focuses on providing a functional system. The aim is to minimize glitches not only because these may affect the safety of the vehicle, but because even minor ones may lead to customer complaints and expensive recalls. Failure analysis and statistical models have proven quite useful at identifying likely areas that may cause problems. This approach, however, does not address interference by malicious adversaries who base their attacks on the exploitation of improbable scenarios (e.g. race conditions, malformed inputs, etc.). Moreover, functionality and safety testing techniques which are used to test the vehicle components are insufficient for catching security vulnerabilities [6], even those which are well-known vulnerabilities, such as buffer overflows [7], which have recently been discovered in the vehicle components and have been exploited [8].

The security posture of the vehicle is further complicated by the fact that is an amalgamation of different software components, written by different teams with widely varying degrees of competence. Safety-critical components comprise very high-quality code and are extensively tested, while others, may simply be off-the-shelf (COTS) code that has been integrated into an ECU (e.g. the entertainment system may be essentially a repurposed personal computer, running an embedded version of Linux [9]). This disparity in the code quality provides the attacker with targets of opportunity as well as the means to gain access to vulnerable software components without the need of physical access to the vehicle [10]. Wireless paths, used not only for V2X communications, but commonplace channels such as Bluetooth, WiFi, cellular, digital radio, even tire-pressure sensors, allow external access to systems on board the vehicle. What is worse, is that these systems primarily involve the off-the-shelf code of mixed quality and large size, usually as a result of poor customization. Hence, such software components can provide limited resistance against a well motivated and trained attacker and to expect otherwise is probably unwise in the face of market realities. The lack of internal security barriers means that having compromised one subsystem, the attacker can end up controlling the entire vehicle (e.g. even stopping of the engine) [11].

https://doi.org/10.24355/dbbs.084-201806251523-0

Therefore, to keep security attacks from spreading across the system, it is imperative to control the communications between system components by defining *who should talk to whom*. By doing that, we provide a level of compartmentalization in the in-vehicle network. With this precondition, a malicious application might remain able to emit (a) malicious packet(s) to its remote peer(s), if it is authorized. But, at the same time, this application can be prevented from attacking other components, which it is not authorized to communicate with. Creating this security policy is already a difficult task, requiring detailed knowledge of possible communication paths between all possible components of the system. When considering an evolving system, which shall be updatable (whereby component interactions may change), this task becomes even more difficult and requires a framework which supports policy generation under concurrent change.

Our approach in addressing this challenge is to build the security policy gradually by integrating it through the design and life cycle of software components. By doing so, the security policy will adapt to changing circumstances during development, integration, and maintenance. Moreover, we will preserve the intentions of the initial designer and ensure the requirements of the actual operational platform. Policy-based communications allow the system to decouple the rules that govern application behavior from the application's functionality, this provides the flexibility to change the application governed behavior without the need to recording system functionality upon changes. We implemented a framework which used to enforce the defined policy based communications to control the different kinds of communications through the internal network of the vehicle.

II. System and threat Model

As described in the introduction, a modern vehicle features a large number of ECUs, which communicate via network (*intra vehicle network*) which we assume to be transparently shared between the ECUs, sensors, and other components. This internal network of the vehicle has already been target of several security attacks [12]. In one example, an attacker intercepted the connection between different ECUs, spoofed the transferred data and emitted false data [10]. In some circumstances, attackers were also able to participate in the communication by fraudulently using the identity of a legit network node. Although ensuring the integrity of exchanged messages between different ECUs inside the vehicle to prevent the message manipulation and replay attacks on the in-vehicle network is already a challenge, still authorization mechanisms are required, which specify permissions and prohibitions to deny malicious ends from performing unprivileged actions or accessing unauthorized resources.

The main challenge in this context is to define a comprehensive access control system to manage communication in the intra vehicle network after the final integration phases in a development process. This challenge originates from (a) the high number of the integrated ECUs in the vehicle and complexity of the communication between the different applications on these ECUs and (b) a late definition of security parameters of the different connections, because the chosen security parameters could violate initial requirements of applications which were specified during the design phase. Moreover, when considering updates of the vehicle system, while yielding the advantage of extending the vehicle's functionality after deployment, updates make the maintenance of the existing access control lists labor intensive and error prone. Each modification of the system, such as an adding new functionality, moving functionality from one ECU to another, or changing security parameters, requires the update of the existing access control lists which can thus lead to configuration errors that could allow unauthorized communication between software components which could in turn be exploited by attackers. In order to prevent the already described effects of unauthorized communication of potentially malicious software components, we propose fine grained access control mechanisms to secure intra- and inter-ECU-communication.

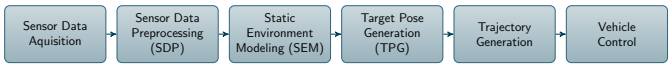

Fig. 1. Exemplary functional processing chain.

a) Use Case: In our exemplary case, we consider the research vehicle MOBILE, built at the Institute of Control Engineering [13], for research in the field of vehicle control and environment perception at the limits of handling. The vehicle is equipped with multiple sensors and computers for environment perception with the goal of providing obstacle avoidance functionality in a static environment on a proving ground.

While the environment perception system is mainly based on LiDAR scanners (based on [14]), a radar sensor and a camera will be added in the future to provide additional information be used to monitor the environment around the car. Data from the LiDAR sensors is used to create a map of the static environment, which provides the basis for model-based trajectory planning. The functional processing chain is displayed in Fig. 1.

In order to generate trajectories for obstacle evasion, the vehicle must acquire and pre-process sensor data (in the case of LiDAR scanners three-dimensional point clouds). Preprocessing is comprised of filter and segmentation algorithms, ensuring that only relevant data is used for modeling the static environment. In the example application, the environment model consists of occupancy grid maps, which provide a map of the drivable and non-drivable areas around the vehicle. From these grid maps, a target pose can be generated for the vehicle. A trajectory towards this target pose is planned through the drivable areas of the environment. This trajectory provides reference values and set-points for the control algorithms which are responsible for maneuvering the vehicle safely through the static environment.

As the environment perception subsystem relies on IP-based Ethernet communication, we base the security model on an according IP stack. Although, we are considering the architecture of a research vehicle, IP-based inter ECU communication has been proposed for automotive purposes in earlier publications (e.g. [15]–[17]), such that the chosen setup can be considered relevant for future automotive applications.

https://doi.org/10.24355/dbbs.084-201806251523-0

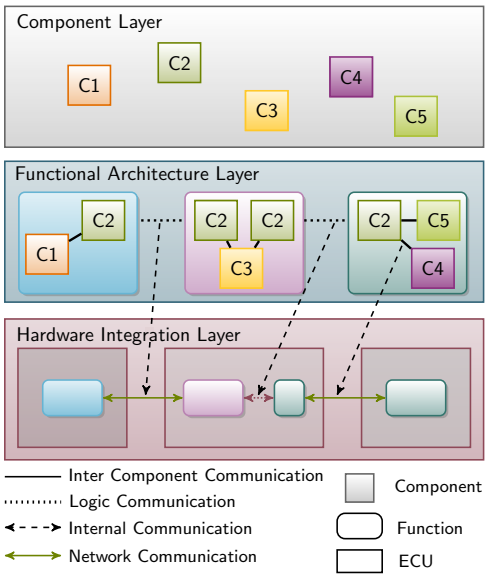

Fig. 2. System layers as also proposed in [18]

III. Policy Framework

A. From Logical Interconnections to Actual Communications

In a component-based system, each functionality in the vehicle is implemented through one or more interacting software components as shown in Fig 2 (cf. [18]). Each atomic component encapsulates different software methods which provide its functionality, behavior, and expose well-defined service interfaces. We assume that a software component can be instantiated multiple times to provide different functionality (e.g. library components for network access). At the same time, functionality can be distributed to multiple ECU's such that software components of the same functionality must communicate over the network (cf. Fig 2).

On the one hand, this composability provides the system integrator with several degrees of freedom for mapping software components of the processing chain to the available ECUs to satisfy different system requirements, e.g. by mapping a component to a remote ECU because of the restricted memory in the local ECU. On the other hand, this makes specifying the communication policy of the components and determining the non-functional concerns (such as real-time constraints, safety, and security requirements) of these defined connections an exhausting operation. Moreover, distributing the functional blocks over several ECUs, due to the assumed transparent communication mechanisms, requires an instantiation of network proxy components during system integration process to enable transparent communication of the network components [18]. These proxy components are usually simple in the sense of their source code complexity. However, they expose additional security weaknesses, particularly as the proxies must respect the both component's requirements.

Due to these facts, we propose to build the security communication policy incrementally by integrating it through the design and life cycle of software components, respectively. By doing so, we guarantee that the components' security requirements are conserved over the different phases (i.e. design, implementation, integration, etc.). Moreover, this allows decoupling the design phase from the integration phase, which is the goal of the project CCC [19], from a security point of view by decoupling communication policy from the components' source code.

To explain how this security framework could be applied, we take the use case of environment perception which was explained in Section II as an example. Fig. 3 shows an extract from the functional processing chain, the required software components as well as a possible distribution of software components over several ECUs for the application at hand.

In the scope of the CCC project, we consider a partially automated V-Model-like development process [19]. Thus we assume that functional requirements are formulated at the beginning of a development process and that security requirements will be formulated during requirement refinement in the implementation process. From functional requirements, a functional system architecture is derived, including services over which the functional blocks need to communicate. For our approach, we also assume that functionality can be mapped to different hardware platforms later in the development process. For this reason, already the coarse functional interfaces must already be attributed with security requirements. The designer can specify some of the link parameters at this stage, such as the security level of the connection, regardless of how an actual implementation might look like.

During implementation, when a software architecture is developed, the generic functional (or logical) interfaces are mapped to actual service interfaces which connect the different software components (cf. Fig. 3). Required and provided services are specified [18] and security requirements, which arise from implementation-specific considerations can be attributed to the service interfaces.

Eventually, when mapping the software architecture to a distributed system of ECUs, additional security requirements arise due to the need to communicate over network. At this design stage, security requirements will again be refined and annotated to the service interfaces.

In our example, the process for the development of the environment perception processing chain would look as follows: The designer of the functionality for *Static Environment Modeling (SEM)* specifies, that the functionality requires a service from *Sensor Data Preprocessing (SDP)* to receive segmented point clouds and provides a environment model over a service to *Target Pose Generation (TPG)* (cf. Fig.3). We use the "$\geq$" expression to indicate that the minimum required protection for this link is integrity.

```
function SEM {
  services{
    provides srv_grid_layer_updated {
     security_level ≥ Integrity;
    }
    requries srv_pc_segmented {
     security_level ≥ Integrity;
    }
  }
}
```

https://doi.org/10.24355/dbbs.084-201806251523-0

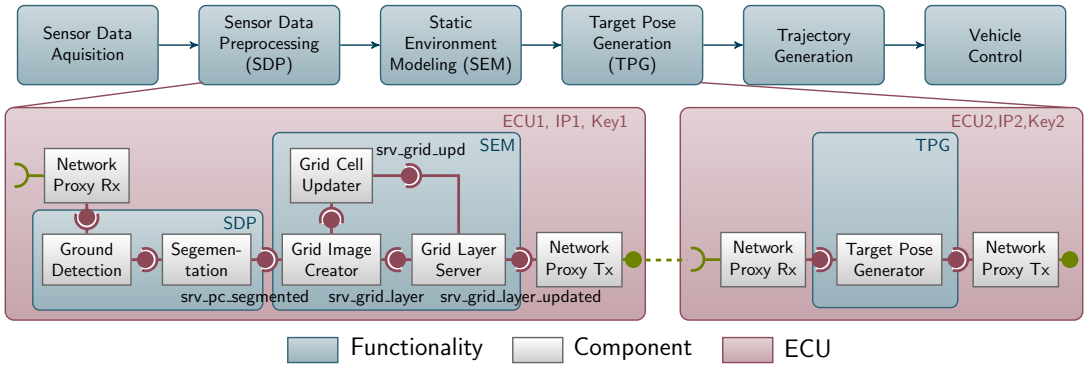

Fig. 3. Software components of use case functionalities and platform mapping

In the same manner, the designer of Target Pose Generation (TPG) functionality defines that the function requires a service ("srv_grid_layer_updated"). It is important to note that at this stage the designer is not concerned about how the two ends of any connection will be mapped and the type of the bus system over which the data will be exchanged. Moreover, the designer could determine the time constraints of the connection by specifying maximum acceptable latencies (e.g. in this case maximum latency of 2 ms), as presented in [20]. After having specified the functional dependencies, the actual implementation of the components can be annotated in the same way, providing information which component provides and requires which services.

```
function TPG {
  component target_pose_generator {
    services {
      requires srv_grid_layer_updated {
       security_level ≥ Integrity;
       timing  2 ground_detection;
      }
    }
  }
}
```

With the implementation at hand, the software architecture can then be mapped to different ECUs in the integration process. The integrator of a specific functionality takes the designer's policy for each component to check the required services for this component and search for the components that provide these services which must be integrated. The integrator can access this information from a local repository which contains all related-platform properties of the integrated components. The integration process may occur in multiple intermediate phases, during each one of these subphases more details could be added to the policy. At the end of these phases, the logical and the inter-component connections which were defined in the design phase are translated to actual communication mechanisms available on the platform. This includes IPC mechanisms or network communication via a network proxy as shown in Fig. 3.

In the given example, the integrator of the components implementing the TPG functionality updates the communication policy between TPG and GLS by adding the actual location (i.e. IP address and ports numbers) of the both components in the platforms. Moreover, the knowledge about the components' mapping gives the integrator the ability to adjust the connection and specify the security protocol which is supported by the link.

```
function TPG {
  component target_pose_generator {
    services {
      requires srv_grid_layer_updated {
        security_level ≥ Integrity;
        security_protocol AH;
        bitrate y;
        ECUs {
          local ECU1;
          remote ECU2;
        }
        IPs {
          local ECU1_IP;
          remote ECU2_IP;
        }
        ports {
          local P1;
          remote P2;
        }
      }
    }
  }
}
```

We note here that the local parameters (e.g. local IP, local port, etc.) refer to the ECU where the request will be evaluated later (i.e. where the component was mapped).

B. *Updating a Component*

In the previous section, we showed how the creation of the security policy could take place in the lab during the production of the vehicle. However, in-field updates are becoming ever more popular as demonstrated by the recent over the air update as Tesla cars and others [19]. Regarding security, the platform should thus ensure the integration of new components without putting the vehicle in a risk of violating the system requirement or creating a vulnerability regarding communication and access control rules. We propose an integration procedure as shown in Fig. 4.

The platform receives a application description package which contains: the new software components, a signed hash

https://doi.org/10.24355/dbbs.084-201806251523-0

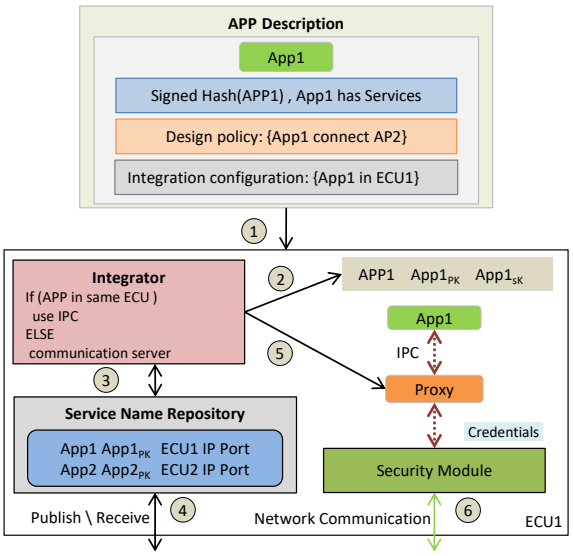

Fig. 4. Communication policy during update an existing component.

value which ensures the authenticity of the application, the design-level credentials, and integration specifications which specify where the component should be mapped. The system's middleware checks the required services that the new component requires (based on the design-level credentials) and determines whether the required application is mapped to the same platform or to a remote platform. The integration component provides an identity for the new application and publishes its information (public key, service name, host ECU, and etc.) via a service name repository. Each ECU contains a service name repository to provide the information about the other services and to broadcast the information of the newly added component to keep the other repositories updated. Regarding the platform where the remote application was mapped, the integration component provides the application with the capability to communicate with local applications or with a proxy with required credentials to use the network.

C. Maintaining Policy Provenance

We have seen how the communication policy can be defined at the designer level in an abstract way, and how policy parameters are modified incrementally during implementation. In such hierarchy, the refined communication policy parameters need to be consistent with existing one in the previous phases, therefore ensuring the provenance of the communication policy during the policy refinement is a crucial goal [21]. We build a trust management module based on a Public Key Infrastructure (PKI) to build trust relations which provide delegation between the multiple entities who participate in developing the communication policy (e.g. a Functionality Designer (F_D), Integrator (I), etc).

Each one of these entities is identified by a public and a secret key. The trust relation is explained in Fig. 5-b, where the trustor entity authorizes a trustee entity under specific conditions. The conditions could be a set of pairs seperated by a logical operations (i.e. &, ||). Each pair represents a defined communication variable and its desired value, operators such as $\leq$, $\geq$, $<$, $>$, $\neq$, or == are used to express the relationship between each variable and its value. The security credential could be derived from the trust relation as follows:

$$
\begin{aligned}
&\text{Trustor: } Trustor_{pk}\\
&\text{Trustee: } Trustee_{pk}\\
&\text{Condition:}\{\ var_1 == val_1\ \&\ var_2 > val_2\}\\
&\text{Signature: Sig with } Trustor_{sk}
\end{aligned}
$$

The security credential which states the public keys of the trustor and trustee as well as the conditions, is signed by the secret key of the trustor entity to ensure the policy integrity The goal of creating such policy is different than the purpose of a certificate [22]. The certificate only authenticates the owners key; it does not prove its trustworthiness.

Fig. 5-a illustrates trust and delegation between the different entities. E.g. in the vehicular domain, an OEM starts the production of a specific vehicle model (e.g. V_{xy}), requiring a set of functionality. Each designer of these functionalities is authorized for providing its functionality. Whenever a connection is required between components in different functionalities, a chain of trust is established between the two functionality designers. Later, the trust is passed to the integrators, applications, and to the platform where the applications are running. By building this trust we creating a chain of trust between the requester and trusted root which is defined in the local policy of the receiver. The receiver authorizes the request if a trust path between the requester and its trusted root can be found.

D. Communication and Policy Enforcement

The distributed nature of a vehicular E/E system's components makes the use of single ECUs for evaluating the communications credentials and enforce the communication policy infeasible. Therefore, we aim to enforce communication policy in a distributed manner by adopting a distributed firewall technique (cf. [23]) so that we equip each ECU with its own security module which acts as a connection ingress policy checker by vetting the incoming and outgoing communication and to enforce the distributed security policy locally. The security module within each ECU contains two main modules: a Communication Module (CM) and Policy Evaluation Module (PEM). CM represents the unique interface which enables the applications to reach the internal vehicle network. However, there is no application component which has a direct access to the CM. All communications need to go through the application's proxy. On the other hand, CM represents the policy enforcement point where it intercepts each incoming and outgoing packet's selectors (i.e. IP, Port, etc.) and applies the relevant security rule (i.e. pass, deny). The PEM is used to evaluate the incoming connection. Its role is limited to the connection setup stage to check the validity of the request.

Connection setup is illustrated in in Fig. 6. We consider that App1, which is located on ECU1, is supposed to initzialize communication with App2, which is assigned to a different ECU (i.e ECU2). Whenever App2 on ECU1 establishes a connection (i.e use a connect socket call), the proxy intercepts this call and redirects it to the communication module. The proxy also delivers all required credentials (Crs) to vouch

https://doi.org/10.24355/dbbs.084-201806251523-0

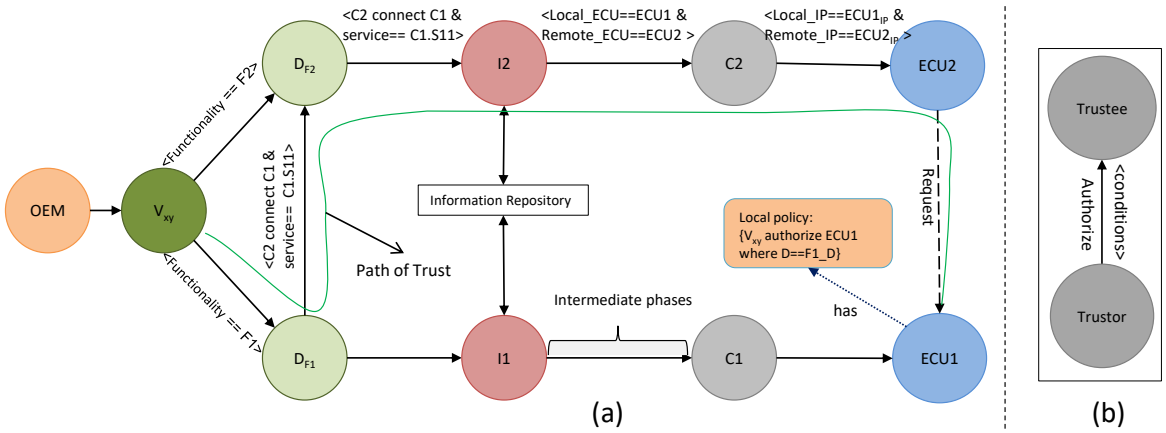

Fig. 5. (a):Trust management module between the different entities. (b) Trust relation between two entities

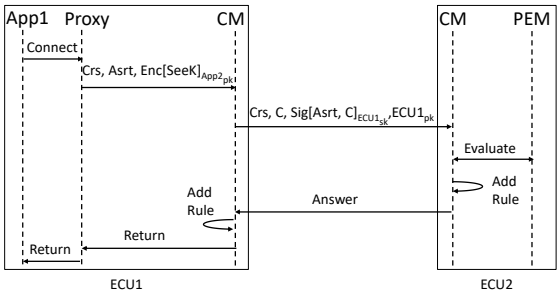

Fig. 6. Initiating a connection and connection setup.

for the local ECU to communicate with the remote ECU and deliver the request. Moreover, it prepares all the required assertions (Asrt) which will be needed by the remote counterpart to evaluate the request. A session key (SeeK) could be chosen by the App1 proxy to later be used as a symmetric key to secure communication between App1 and App2 after validating the request. This key will be changed at a later point in regular intervals. This key is encrypted by the remote application public key ($App2_{pk}$) to prevent any third party from perceiving it during the request transmission.

CM on ECU1 uses the private key ($ECU1_{sk}$) ECU1 to sign the received assertion alongside with a monotonic counter value (C) to ensure communication integrity. This monotonic counter value (C) is used to guarantee that messages are up-to-date and to prevent replay attacks. The signed assertion with the timestamp, the ECU's public key ($ECU1_{pk}$) and the credentials are all sent to the remote communication counterpart (i.e. ECU2). The communication module in the ECU2 deliver the received request and the credentials to the PEM to evaluate them based on the local communication policy, If the request is authorized by ECU2's PEM, a new security rule will be added to the CM. This rule includes all the communication identifiers such as the local and remote IP, local and remote applications port, remote public key, the required security service (i.e. integrity, confidentiality), access control role (i.e. authorize, deny), an identifier for the session key, and others. Later, CM on ECU2 sends the answer of the request to the remote CM. In case of authorized connections, a similar rule is added to the communication module of ECU1.

IV. Evaluation

a) Implementation: For implementation of the properties described in the last section, we apply the Keynote policy definition language [24] for creating credentials and security policies, as well as to build relationships that provide authorization of security actions. The security module which is used to evaluate the defined credentials and policies are implemented with in a Genode microkernel OS [25]. The use of Genode allows us to leverage its efficient message-passing capabilities for the enforcement of the socket APIs of the different application, by replacing direct calls with inter-process communications calls (IPC) that are implemented via the Genode message-passing mechanism. The implemented proxy is responsible to translate the socket calls to the proper IPC call and redirect it either to the CM or to a local application in the same platform. Proxy provides an efficient and flexible access control layer to enforce the security policy between the applications [26]. The CM contains light weight TCP/IP network stack (LWIP) and embedded IPsec protocol [27] which is used mainly to provide the communication integrity. The Keynote library is used to implement the PEM.

b) Evaluation: To evaluate our policy scheme, we used a prototype implementation of the security framework to measure the introduced overhead of credential verification. We evaluated our module regarding its overhead in terms of required memory and latency. Our test platform is comprised of two Raspberry PI 2 model B. Both platforms are running Genode OS and our framework, communicating over a local network. We used a Raspberry Pi (RPi) platform to show adaptability of our framework to an embedded platform with limited processing power and memory.

Regarding latency, we measure the average elapsed time to initiate a connection between the two RPis using our communication scheme and compare it to the needed time to connect with the absence of policy decision module. The introduced overhead is introduced from two parts: the necessary time to transmit the required credentials and time overhead of the request evaluation. At the same time, we want to show the effect of the number of the credentials to evaluate on transmission and request evaluation times. In this respect, Fig. 7 shows the overhead observed from transmitting and evaluating the credentials, it shows the base overhead

https://doi.org/10.24355/dbbs.084-201806251523-0

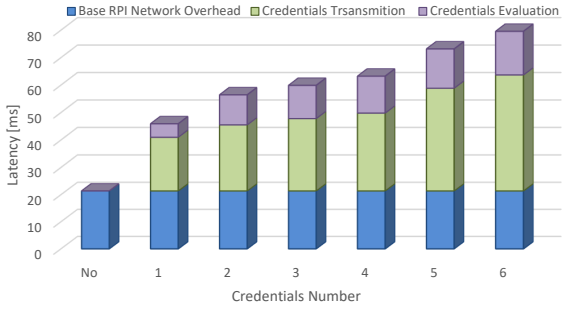

Fig. 7. Performance evaluation of the security module.

TABLE I
COMPONENTS SIZE

component	size KB
1 credential	1.2
Public Key	0.38
Private Key	1.6
Policy	0.5
signed request	0.8

of the network stcak on the RPi without the use of our security module. Each time we used a different number of credetinals (i.e. from 1 to 6) to evaluate the performance. We could notice that increasing the number of delegations (i.e. credentials numbers) introduces more overhead in both parts (i.e. evaluation and transmition).

Other than that, we evaluated the overhead in size for the transmitted messages. Table I defines the size for each required file to set up the connections (i.e. credentials, public key, signed request). From the same table, we can estimate the required saving size for each application within 1-level of delegation which is around 3.7 KB (i.e. the size of credential, keys). we can estimate the size of the transited data for the evaluations which is also around 2.9 KB (i.e. credential, public key, signed request).

One important aspect to notice at this point is that network performance of the available network stack on the hardware platform already introduces performance issues which is illustrated in Fig. 7. This related to the USB driver used with RPI and the way of processing the high interrupt load on the system which interferes with the network stack [27]. Most of the evaluation overhead comes from the signature verification of the credentials; we can lessen this overhead by caching the verified credentials and used it later with out the need to verify its signature again.

V. Related Work

The security issues in the intra vehicle network was the topic for many research [28], [29]. Many authors indicate the need for access control framework to control the communications of the intra vehicle network [30], [31]. However, few solutions have been suggested. Wolf et al. [32] proposed the use of a centralized gateway to interconnect the different existing bus systems in the vehicle. This gateway was used as a firewall to intercept the exchanged communication. Such a gateway could be also used to ensure the valid behavior of exchanged message regarding the time constraints based on a given policy as in [33]. Using a central security gateway could protect the intra vehicle network, while having it some limitations: Firstly, gateway provides an attack surface which may be considered as a single point of failure. Moreover, the internal communication inside each subnet is still not controlled. Chutorash [34] proposes an approach for using a firewall to control the interaction in the vehicle communication. The approach is restricted to monitoring the interaction between human-machine interface systems and other vehicle components, which ignored controlling the interaction between the vehicles components. In both previous frameworks, the mechanism to define the access control rules of the firewall is not discussed. Zrelli et al. [35] implemented access control rules at both the data link layer and the network layer.

Blaze et al. [36] invented the concept of Trust Management to solve access control problems in distributed systems. They proposed a language (i.e. Keynote language [24]) to express the access control rule the security policy. However, many other languages serve the same purpose such as XACML [37] and SPL [38]. At the same time, KeyNote provides several advantages, such as the delegation mechanism. Many authors adopt the trust management concept to implement an access control framework in many fields, such as web applications [39], distributed firewall [23], and others.

VI. Discussion and Conclusion

Our work is based on the observation that controlling the communications between different applications within a vehicular platform is a necessary prerequisite in containing an attack and preserving the security of the intra-vehicle network. However, defining a static communications policy for a vehicular distributed system is both labor-intensive and error-prone and it becomes more difficult if it is performed during the final integration. We proposed a methodology supporting a *gradual* definition of the security policy, starting with the designer of the component, and then adapting and specializing the policy as the component is linked to other components and is integrated with the rest of the system. Additionally, we created a framework which allows communication policy to be deployed incrementally and ensures integrity of this policy during the life cycle of software components.

The advantage of using a policy definition language for the expression of the vehicular communications policy is that we have a lot of expressive power which is reflected in our ability to *refine* and *extend* the original component policy during integration while ensuring that the adaptations are consistent with the original policy. By creating a “chain of trust” from the original designer to the production system, we ensure the consistent application of the security policy without the need for manual intervention

Maintaining the security policy enforcement mechanism on the operational vehicle does not impose excessive performance burden as demonstrated by the results from the preceding section. There we noted that the credential evaluation, whose overhead is shown in Fig. 7, is done *once* per link setup. This operation usually takes place during power on, although, caching of policy may allow results from previous configurations to be reused, further reducing the effort. Credential

https://doi.org/10.24355/dbbs.084-201806251523-0

evaluation is not repeated, unless the system is reconfigured. Moreover, re-keying the session key, which is done at regular intervals, does not require a credential evaluation operation. Even in cases where during operation, a vehicle may face a failure and need to switch from a primary system to a backup system, the necessary policy may be already in place. We expect that reconfiguration will be done only when the vehicle is quiescent.

Once the secure channels are setup, the overheads are similar to the statically configured case. Moreover, since we are more concerned about mutual authentication and integrity, rather than confidentiality (although the policy caters for that, if needed), most of the communications will not involve encryption and will present a much lighter load to the system.

Finally, we address the issue of key storage and signing operations through the use of a dedicated crypto module, which will be responsible for all crypto operations and for storing the keys, or by placing the crypto code in a trusted environment, e.g. the ARM TrustZone technology.

ACKNOWLEDGEMENT

This work is supported by the DFG Research Unit Controlling Concurrent Change (CCC) project, funding number FOR 1800. Additionally, it is supported by the European Commission through project H2020 ICT-32-2014 SHARCS under Grant Agreement No. 644571.

REFERENCES

[1] C. Englund, L. Chen, A. Vinel, and S. Y. Lin, "Future applications of vanets," in *Vehicular ad hoc Networks*. Springer, 2015, pp. 525–544.

[2] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE 06. New York, NY, USA: ACM, 2006, pp. 33–42.

[3] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

[4] R. Charette, "This car runs on code," feb 2009. [Online]. Available: http://www.spectrum.ieee.org/feb09/7649

[5] Y. Laarouchi, Y. Deswarte, D. Powell, J. Arlat, and E. De Nadai, "Ensuring safety and security for avionics: A case study."

[6] S. Bayer, T. Enderle, D.-K. Oka, and M. Wolf, "Automotive security testingthe digital crash test," in *Energy Consumption and Autonomous Driving*. Springer, 2016, pp. 13–22.

[7] J. P. Anderson, "Computer security technology planning study. volume 2," DTIC Document, Tech. Rep., 1972.

[8] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, 2011.

[9] D. YONEY, "Tesla model s owners hack their cars, find ubuntu," Apr 2014. [Online]. Available: https://www.autoblog.com/2014/04/12/tesla-model-s-owners-hack-their-cars-find-ubuntu/

[10] R. M. Ishtiaq Roufa, H. Mustafaa, S. O. Travis Taylora, W. Xua, M. Gruteserb, W. Trappeb, and I. Seskarb, "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study."

[11] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *In Proceedings of IEEE Symposium on Security and Privacy in*, 2010.

[12] M. Hamad, M. Nolte, and V. Prevelakis, "Towards comprehensive threat modeling for vehicles," in *the 1st Workshop on Security and Dependability of Critical Embedded Real-Time Systems*, 2016, p. 31.

[13] P. Bergmiller, "Towards Functional Safety in Drive-by-Wire Vehicles," Ph.D. dissertation, 2014.

[14] J. Rieken, R. Matthaei, and M. Maurer, "Toward perception-driven urban environment modeling for automated road vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015.

[15] M. Glass, D. Herrscher, H. Meier, P. Schoo *et al.*, "Seis security in embedded ip-based systems," *ATZelektronik worldwide*, 2010.

[16] A. Kern, "Ethernet and ip for automotive e/e-architectures-technology analysis, migration concepts and infrastructure," Ph.D. dissertation, Erlangen, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2012.

[17] A. Bouard, B. Glas, A. Jentzsch, A. Kiening, T. Kittel, and B. Weyl, "Driving automotive middleware towards a secure ip-based future," *Proc. of ESCAR*, 2012.

[18] J. Schlatow, M. Nolte, M. Möstl, I. Jatzkowski, R. Ernst, and M. Maurer, "Towards model-based integration of component-based automotive software systems," in *Annual Conference of the IEEE Industrial Electronics Society (IECON17)*, Beijing, China, 2017.

[19] A. Reschka, M. Nolte, T. Stolte, J. Schlatow, R. Ernst, and M. Maurer, "Specifying a middleware for distributed embedded vehicle control systems," in *Proceedings of the 2014 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Hyderabad, India, December 2014, pp. 117–122. [Online]. Available: http://dx.doi.org/10.1109/ICVES.2014.7063734

[20] S. Holthusen, S. Quinton, I. Schaefer, J. Schlatow, and M. Wegner, "Using multi-viewpoint contracts for negotiation of embedded software updates," *EPTCS 208*, p. 31.

[21] J. Cheney, "A formal framework for provenance security," in *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*, 2011.

[22] R. Housley, T. Polk, D. W. S. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280, 2002.

[23] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a distributed firewall," in *Proceedings of the 7th ACM conference on Computer and communications security*, 2000.

[24] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The keynote trust-management system version 2," RFC 2704, 1999.

[25] Genode OS framework. [Online]. Available: https://genode.org/

[26] T. Jaeger, K. Elphinstone, J. Liedtke, V. Panteleenko, and Y. Park, "Flexible access control using ipc redirection," in *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, 1999.

[27] M. Hamad and V. Prevelakis, "Implementation and performance evaluation of embedded ipsec in microkernel os," in *Computer Networks and Information Security (WSCNIS), 2015 World Symposium on*, 2015.

[28] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 528–533.

[29] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," in *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, 2013.

[30] H. Bar-El, "Intra-vehicle information security framework," in *ESCAR*, 2009.

[31] H. Schweppe and Y. Roudier, "Security and privacy for in-vehicle networks," in *Vehicular Communications, Sensing, and Computing (VCSC), 2012 IEEE 1st International Workshop on*. IEEE, 2012, pp. 12–17.

[32] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Workshop on Embedded Security in Cars (ESCAR)*, 2004.

[33] S. Seifert and R. Obermaisser, "Secure automotive gatewaysecure communication for future cars," in *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*. IEEE, 2014, pp. 213–220.

[34] R. Chutorash, "Firewall for vehicle communication bus," Feb. 24 2000, wO Patent App. PCT/US1999/017,852. [Online]. Available: http://www.google.de/patents/WO2000009363A1?cl=en

[35] S. Zrelli, A. Miyaji, Y. Shinoda, and T. Ernst, "Security and access control for vehicular communications," in *Networking and Communications, 2008. WIMOB'08. IEEE International Conference on Wireless and Mobile Computing,*. IEEE, 2008, pp. 561–566.

[36] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*.

[37] extensible access control markup language(XACML). [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[38] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes, "Spl: An access control language for security policies and complex constraints."

[39] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss, "Referee: Trust management for web applications," *Computer Networks and ISDN systems*, vol. 29, no. 8-13, pp. 953–964, 1997.